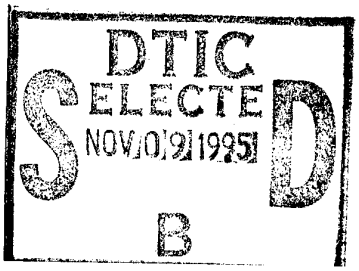
AR-009-200
DSTO-GD-0043



A Windows Program for Multivariate Optimization

David R. Skinner

19951108 099

Approved for Public Release

DTIC QUALITY INSPECTED 5

© Commonwealth of Australia

DEPARTMENT OF DEFENCE
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

THE UNITED STATES NATIONAL
TECHNICAL INFORMATION SERVICE
IS AUTHORISED TO
REPRODUCE AND SELL THIS REPORT

A Windows Program for Multivariate Optimization

David R. Skinner

Maritime Operations Division
Aeronautical and Maritime Research Laboratory

DSTO-GD-0043

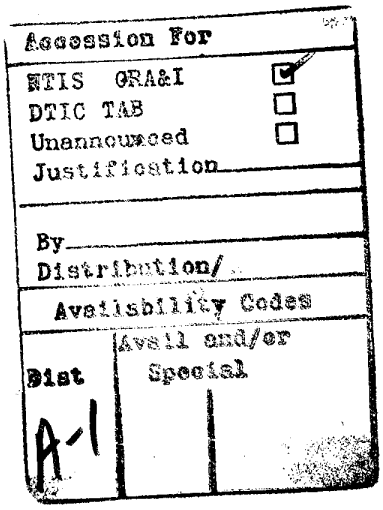
ABSTRACT

A program is described that implements the "centroid" algorithm for the optimization of functions of many variables by a stochastic direct-search procedure. The program is available as a Windows application with context-sensitive on-line help, and can maximize any suitable function exported by a dynamic-link library (DLL). For practical reasons of simplicity, the program is limited to real functions of up to 20 real variables, and the search domain is limited to hyper-rectangular blocks in multi-dimensional Cartesian spaces. The only constraint on the function to be optimized is that it should be single-valued at every point in the domain, and should be capable of being evaluated in a DLL. It need not be smooth or continuous. A sample program is provided to illustrate the generation of a suitable DLL using Borland Pascal. In general, the algorithm employed will, given a sufficient number of search iterations, find the global maximum of a function in the presence of closely competing local maxima.

Approved for public release

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION



Published by

DSTO Aeronautical and Maritime Research Laboratory
PO Box 4331
Melbourne Victoria 3001 Australia

Telephone: (03) 9626 7000
Fax: (03) 9626 7999
© Commonwealth of Australia 1994
AR No. 009-200
June 1995

APPROVED FOR PUBLIC RELEASE

A Windows Program for Multivariate Optimization

EXECUTIVE SUMMARY

A recent paper by the author and a colleague (Benke and Skinner, 1991) describes a new method, the "centroid" algorithm, for finding the maximum value within a given domain of a function of many variables. This method is based on a theory of noise reduction, and is applicable to any type of function that is single-valued and finite at every point in the domain, the only restriction being that it should be capable of being calculated in a computer program. Being a statistical process, the method does not require the function to be either analytically or numerically differentiable, or to have only a single peak in its domain, but unlike most statistical algorithms does not need any arbitrary scaling or "cooling" parameters. Finally, the method is relatively insensitive to the number of variables, unlike Monte Carlo techniques where computing effort varies exponentially with this factor.

The referenced paper describes a number of test applications of the centroid algorithm, including a function subject to random fluctuations much greater than the underlying variance, and a 13-dimensional function where advantage was taken of symmetries in the function definition. Another problem successfully solved elsewhere involved a simple function of 100 000 variables. However, it was felt that there was a need for a general-purpose implementation of the algorithm for convenient solution of the most common case, a function of a few (up to 20) variables in a continuous Cartesian space, with no advantage taken of symmetries. That is what this program implements.

The approach taken was to require the user to program a dynamic-link library (DLL) with the function to be optimised. Given a suitable language compiler, which includes at least some versions of Pascal, C++, Fortran and Common Lisp, this is a relatively simple operation; in Borland Pascal version 7.0 it requires only five lines of code additional to the definitions of the function and the type of variable domain. This approach was preferred to the provision of build-in functions and interpretation of equations entered at run-time, since it gives much greater flexibility, and users of multi-dimensional optimisation are very likely to be proficient in at least one of the suitable languages. The report contains an example of a suitable DLL, which optimises a function used in Benke and Skinner (1991).

The program is implemented in Windows, with an interface of a type that will be familiar to most users. A dialog box is used to select, by bringing up a standard file-open dialog, a DLL for the function to be optimised, and to enter the name of the function and the number of variables. This causes a table for minimum and maximum values to expand to accommodate all the variables. When the table has been filled in, together with optional values for the number of iterations of the algorithm and for a random-number seed, the optimisation can be run any number of times. While it is running, there is a continuously changing display of the number of iterations completed and the current best estimates for the optimised function and its position in the variable domain. It is possible to switch between function DLLs, or between functions in a DLL, during a single session with the program. The program uses a standard form of Windows context-sensitive on-line help, with the usual hypertext links and pop-up definitions.

Contents

1. INTRODUCTION 1

2. DOCUMENT CONVENTIONS 1

3. DESCRIPTION OF THE PROGRAM 2
3.1. Hardware and Software Requirements 2
3.2 Program Files and Sub-directories 2

4. WRITING THE MERIT FUNCTION 3

5. RUNNING THE PROGRAM 4
5.1 The Set Up Option 4
5.2. The Run Option 6
5.3 Leaving the program 7

6. WHAT TYPES OF FUNCTION CAN BE OPTIMIZED? 7

7. SUMMARY 8

REFERENCE 8

Figure 1. Parameter set-up dialog box in initial state. 9
Figure 2. Parameter set-up dialog box after data entry. 9
Figure 3. Progress-report window in initial state. 10
Figure 4. Progress-report window after start of optimisation 11

APPENDIX A Description of the Centroid Algorithm 13

APPENDIX B Sample Source Code for Dynamic-link Library 15

1. Introduction

This report describes a Windows application program for the optimization of a multimodal function in many dimensions, and provides a sample, in Borland Pascal, to illustrate how a dynamic-link library (DLL) may be written to define the function. The "centroid" algorithm used is described by Benke and Skinner (1991), and briefly summarized in Appendix A. Essentially, the algorithm involves repeated generation of random sets of parameters, within the constraints of the problem, and produces weighted averages (centroids) in a way designed, according to a simple theory of noise reduction, to produce a set that minimizes the deviation from the optimum. As described in the reference, the algorithm has been applied successfully to a number of problems varying from simple two-dimensional functions to complicated functions of several hundred variables, although this implementation is limited to 20 variables. For reasons of simplicity in defining the function, no use is made of the potential reduction in computing effort that could be achieved by taking advantage of symmetries of the function in the search domain.

An obvious question is, how can the program possibly know how to do anything, when it has no information on the problem? The answer is that it provides the basic algorithm and all the necessary housekeeping, and it is up to the user to provide one or more DLLs, each defining one or more functions to be maximised. For many problems, writing the function is not a particularly onerous chore, and an example is given in the source code provided in Appendix B

2. Document Conventions

Where an instruction refers to keys joined by + signs, these are to be entered simultaneously. Where references to keys are joined by commas, they are to be entered consecutively. Thus

enter Alt+F, S

means press the F key while the Alt key is held down, release both keys and then press the S key. The use of capital letters here does not imply pressing the Shift key. When an instruction refers to menu selections separated by commas, each is contained in a sub-menu of the previous. Thus

select File, Set Up from the menu system

means use the mouse to click the word File on the main (permanently displayed) menu, then click the words Set Up on the sub-menu that will appear in response to the first click.

Fixed-pitch type, such as **centroid.exe**, is used where it is necessary to refer to program file names, or to identifiers within program files.

3. Description of the Program

3.1. Hardware and Software Requirements

The program requires to be run under Windows 3.0 (or later), and the user must have a language compiler capable of generating DLLs (this includes at least some implementations of Pascal, C++, Fortran and Common Lisp). The program will run much faster if the computer is equipped with an arithmetic co-processor.

3.2 Program Files and Sub-Directories

The program is provided on a distribution disk, copies of which are available on request from the Chief, Maritime Operations Division, at the address on the title page. It takes the form of six files, which are:

(a) **centroid.exe**. This is essentially just a user interface, written in Visual Basic. It is used to input library and function names, and other data relevant to an optimisation. It also maintains a watching brief on the optimisation, and presents continually revised data on progress.

(b) **cent.dll**. This provides the algorithm, together with its housekeeping and interaction with the Windows Application Program Interface. It was written in Borland Pascal.

(c) **centroid.hlp**. This provides standard Windows on-line help, with the usual pop-up definitions, cross-referenced jumps and search facilities. The help is context-sensitive, that is, at any time the user can select a Help button or menu item, or press the F1 key, and be presented with the help topic most likely to be relevant to the current state of the program. It was written as a rich-text format (rtf) file, using Microsoft Word, and converted to a help file using the Microsoft help-file compiler (**hcp.exe**).

(d) **cmdialog.vbx**, **commdlg.dll**, **threed.vbx**. These are standard Visual Basic DLLs that are used for generating various dialog boxes and for accessing the help file. (The copyright to these files is held by the Microsoft Corporation, but distribution with applications written in Visual Basic is permissible.)

To instal the program automatically, insert the distribution disk, select File, Run from the Windows Program Manager menu system, and type

a:\setup

in the space labelled Command Line. If the disk is not in drive A, change the first character of the command accordingly. The set-up program will ask for a sub-directory name for the installation, and will suggest **c:\centroid** as a default. It will copy the two main files, **centroid.exe** and **centroid.hlp**, into this, and the remaining files into a sub-directory determined by your Windows set-up (if they are not already there). Finally, it will generate a group containing icons for the program and its help file.

If Windows is running from a network and the sub-directory for the DLLs is write-protected, the automatic set-up program cannot be used. In this case, or if you want to control over-writing of existing DLL files, the files may be copied manually from the **a:\manual** sub-directory to any sub-directories of your choice, bearing in mind that the help system expects to find **centroid.hlp** in the same sub-directory as **centroid.exe**, and that the remaining files should be in the default library-search path. You will then need to generate one or two program items, and optionally a program group, using the techniques described in the Windows manual. During manual set-up, you can, if you wish, change the main part of the file names of the **exe** and **hlp** files provided these remain the same as each other, but the remaining four files must retain their original names.

If the Browse button (see later) is used for finding DLLs, a small file named **centroid.ini** is written into the Windows directory. If you have changed file names during manual set-up, this file will have a corresponding change. The file enables the browse algorithm to start looking in the sub-directory where the last DLL was found, even if this was during a previous session with the program.

4. Writing the Merit Function

In order to find the maximum value of a function, it is necessary to provide a dynamic-link library defining the function. The DLL is dynamically linked to the main program at run time, which means that the user can switch between libraries, or functions within the libraries, to perform a number of optimisations in one session. The name of the library within the file is irrelevant.

The merit function may be programmed in any language that supports DLLs. It must return a real value expressed in the IEEE four-byte floating-point standard, which has various names in these languages, as shown in Table 1. The only parameter to the function should be a contiguous group of the same type of variable, called by reference. This may be an array, or any other type of record or structure as defined for the language in use - the example in Appendix B has a Pascal user-defined **record** type. Multi-dimensional arrays are not recommended, since the order in which their elements are assumed to be stored in memory depends on the language used.

Table 1. Names of Real Types in Various Languages

Language	Name of Four-Byte Real
Pascal	`single`
Fortran	`real*4`
C++	`float`
Common Lisp	`float`

The program can recognise a function either by its name, which is not case sensitive, or by its ordinal number. The techniques for assigning exported names and ordinal numbers to functions in DLLs are described in the manuals for the various languages. The example in Appendix B shows how an ordinal number is assigned in Borland Pascal - it is the number following the word **`index`** in the **`exports`** statement. In this case, no exported name is specified, so it will default to the identifier (**`MultiGaussian`**) used in the code.

In Borland Pascal, it is necessary to use the {**`$N+,E+`**} compiler directives to access the **`single`** type and to allow the program to be run on a machine without a mathematics co-processor. It is also desirable, in the interests of generating readable code, to define a **`record`** type (rather than an **`array`** type) for the parameter, and to use a **`with`** clause in the merit function. Appendix B, which implements the multi-Gaussian function of the reference, demonstrates the form of a suitable Pascal DLL. It can be seen from this example that (apart from non-functional comments and blank lines) the source code contains only the definition of the parameter set, the function to be optimised and five additional lines. These are the **`library`** statement, a compiler-directive comment, an **`exports`** statement and the two lines of an empty initialisation block.

5. Running the Program

When the program is started, the user is faced with a blank desk-top and a menu containing two items, File and Help. The Help item gives access to a standard Windows help file that is an abbreviated version of this report. The File item contains three options, labelled Set Up, Run and Exit, with the Run option initially disabled.

5.1 The Set Up Option

The program starts off with no information about what function is to be optimised, about the number of values in the parameter set passed to the DLL, or about the

domains that these values are drawn from. In order to run an optimisation, therefore, it is necessary to set up a number of conditions. To do this, you can

select File, Set Up from the menu system,
enter Alt+F, S,
enter Ctrl+S, or
press the Return key.

This will bring up the parameter set-up dialog which is initially as shown in Fig. 1. The top-left section (dark blue text on most screens) contains edit boxes for the library and function names, the number of parameters, a random-number seed and the maximum number of iterations for the algorithm. The top-right section (brown or dark red text on most screens), has edit boxes for the minimum and maximum values for all parameters. Initially, there is room for only one parameter. The third section contains buttons labelled Accept, Cancel and Help.

To set the function for optimisation, first enter the library name in the top edit box, which initially displays `*.dll`. It is not necessary to include the extension if this is `.dll`. If no path is entered, the program carries out a default library search (for more details, see the on-line help). Alternatively, you can select the Browse button (or enter Alt+B) to search for the DLL file using a standard file/load dialog. This has the advantage that you will not need to type in the file name, or you may indeed have forgotten the name, but you will still need to remember either the function name or its ordinal number, and will need to know the number of parameters and the domains from which they are drawn. Once the browse facility has been used, the dialog box will always start in the sub-directory from which the last DLL was read, even if the program has since been closed. In the second edit box, you can enter either the ordinal number or the name of the function - the default is ordinal number 1.

To set the number of parameters, simply edit the third box, and the number of lines in the second section will change to accommodate the value entered. If your entry cannot be interpreted as an integer in the range 1 to 20, an error message will appear.

There is no need to change the random-number seed, which is initially zero. However, changing this results in a different set of pseudo-random parameter values, to allow for different starting points on the random walk through parameter space. Any value entered here must be capable of interpretation as a long integer, i.e. a standard four-byte signed integer.

The maximum number of iterations is initially set at 1000. This is usually adequate for a moderately complicated function of a few variables, but it can be edited to any number from 1 to the maximum for a long integer. Note that this is a maximum only - iteration can be stopped at any time without loss of information.

After the number of parameters has been selected, minima and maxima for these can be entered in the appropriate edit boxes, in the same order that they are defined in the DLL. These entries must be capable of interpretation as `single` values. The random

parameter values will be selected uniformly from these ranges, so the parameter domain is effectively a hyper-rectangular solid in a multi-dimensional Cartesian space.

A completely filled-in set-up dialog, for the example in Appendix B, is shown in Fig. 2. The use of the function name in the second edit box is redundant and purely for demonstration - the default ordinal value 1 could have been left untouched.

After the dialog box is completely filled in, the user may select either the Accept or Cancel button. The Accept button may be selected from the keyboard using Alt+A or Enter, and the Cancel by Alt+C or Escape. As is conventional, the Accept button changes the program values to those in the dialog box, and the Cancel buttons clears the dialog box without making any changes.

5.2. The Run Option

To run an optimisation,

select File, Run from the menu system,
enter Alt+F, R, or
enter Ctrl+R.

This cannot be selected until you have accepted the set-up dialog box at least once. When it is selected, it brings up the progress-report window, which, when there are two parameters as in Appendix B, is initially as shown in Fig. 3.

Start the optimisation by selecting the Start button (or Alt+S or Enter). The window labelled Iteration will display the number of times the program has cycled through the algorithm, until either you select the Stop button (or Alt+P or Escape), or the iteration count reaches the maximum set in the set-up process. The window labelled Parameter Values will display the current best estimates for all the parameter values, changing as the estimates improve.

The window labelled Summary of Last Ten Changes shows the iteration count each time the current best estimate for the parameter set improves, the function value at each change, and the algorithm status for that change. This latter value can be Initial (black text on most screens) if the parameters are the initial random set, Learned (dark green text) if the set resulted from successful application of the algorithm, or Random (red text) if the last random parameter set produced the current best estimate. Usually, after an initial settling-in period, the status will be overwhelmingly Learned. Figure 4 shows the progress-report window shortly after the start of an optimisation.

After the optimisation is complete, you can either re-select Start or select Exit (or Alt+X). Re-selecting Start causes a new run starting from scratch, but with the random-number generator running on from its last call. This will give a new random walk through parameter space, but with the original values for parameter limits and

maximum number of iterations. Selecting Exit allows you to run the set up again, or to exit the program.

5.3 Leaving the program

If either the set-up or progress-report window is displayed, select Cancel or Exit as appropriate. When you can see only the main window,

select File, Exit from the menu system,
enter Alt+F, X,
enter Alt+F4, or
press the Escape key.

You can also leave by using the system menu box to close the main window.

6. What Types of Function can be Optimized?

The function to be optimized needs to be capable of being evaluated in a computer program, and to be single valued at all points in the search domain, otherwise there are no restrictions. Within these constraints, the function need not be analytic, smooth or continuous - one problem solved by Benke and Skinner (1991) used a function subject to random noise.

Because the optimization is a stochastic process, it is not possible to analyze its behaviour for different merit functions (although some theory is given in the reference). It is possible, however, to demonstrate that the process will behave like an inefficient Monte-Carlo search if the maximum value of the merit function is close to zero in a domain that spans zero - the analysis is restricted to functions that are everywhere positive and finite. If there is a problem with slow convergence, it is suggested that the user try to re-state the merit function so that (a) it is everywhere positive, and (b) there is a large variation of merit function over the search domain, since, contrary to what one might intuitively expect, convergence can be fastest when the global peak occupies a relatively small part of the domain. There is some suggestion, from observation of various types of function, that convergence is best for isotropic parameter sets, that is, sets in which all parameters perform similar functions and all are of comparable size.

7. Summary

This report describes the practical implementation, as a Windows application with context-sensitive on-line help, of the centroid algorithm defined by Benke and Skinner (1991). It is presented in a form such that the user need only provide a function definition in a dynamic-link library. An example is provided to illustrate how this is achieved in Borland Pascal.

Reference

K.K. Benke and D.R. Skinner, *A direct search algorithm for global optimization of multivariate functions*, **Australian Computer** J., Vol. 23, no 2, pp 73-85, May, 1991.

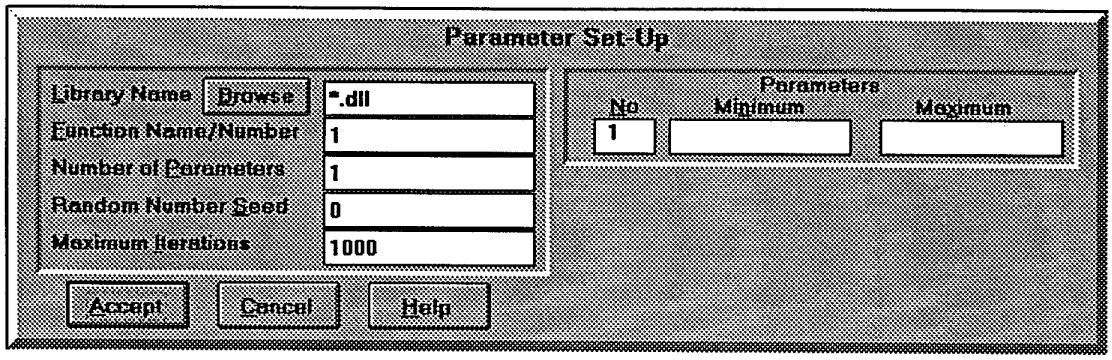


Figure 1. Parameter set-up dialog box in initial state.

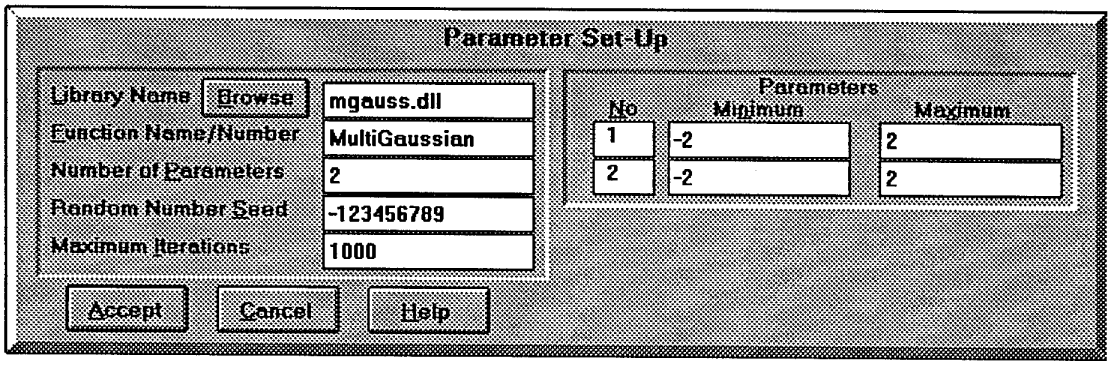


Figure 2. Parameter set-up dialog box after data entry.

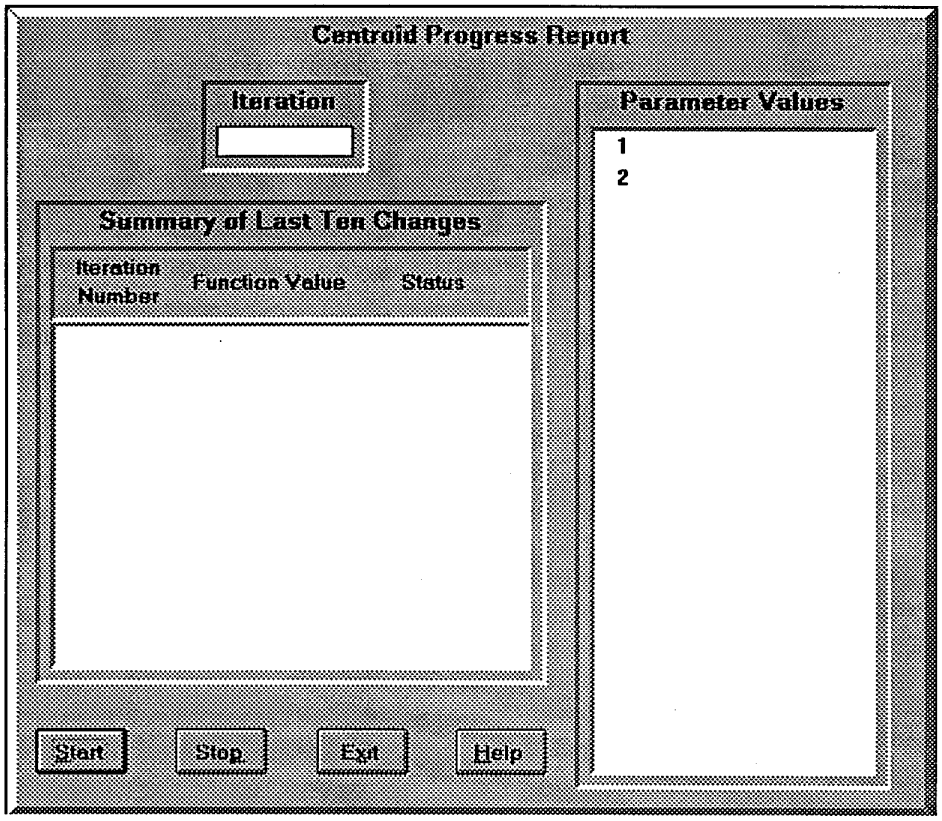


Figure 3. Progress-report window in initial state.

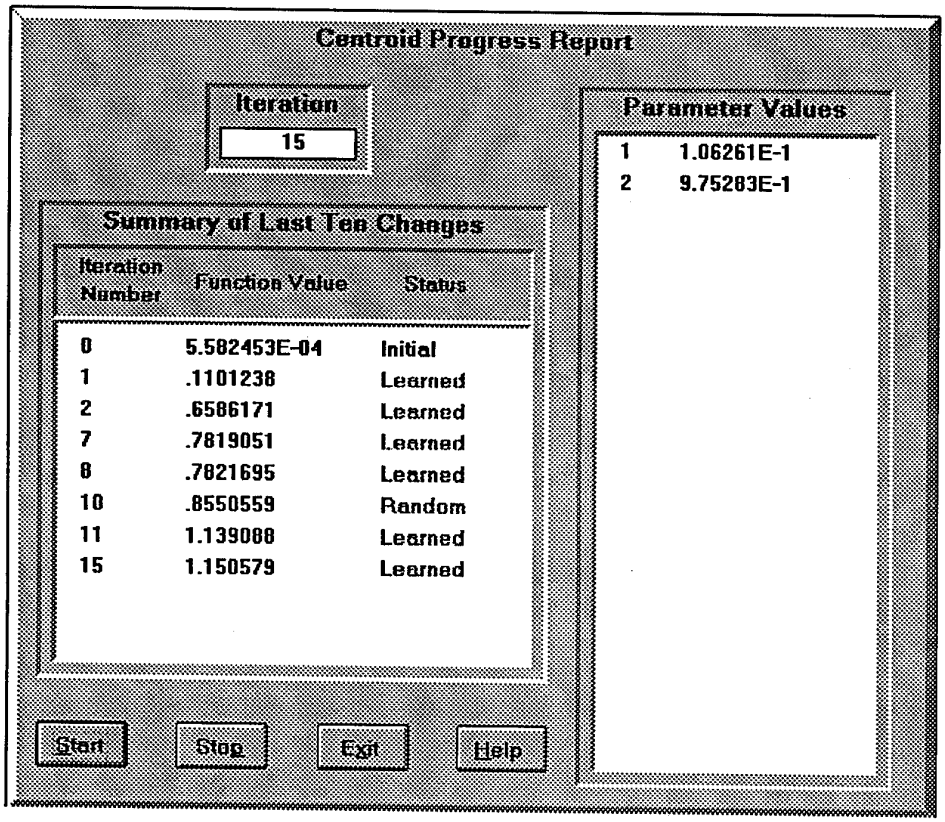


Figure 4. Progress-report window after start of optimisation

This page intentionally left blank

Appendix A

Description of the Centroid Algorithm

This Appendix represents a very brief summary of the reference. The problem is, given a merit function, $f(\mathbf{X})$, of a set of parameters, represented as a vector $\mathbf{X}$, to find the value $\mathbf{X}_{opt}$, within a given domain, for which the function is maximum. We consider only functions that are everywhere positive and finite.

We can consider any vector $\mathbf{X}$ as the sum of $\mathbf{X}_{opt}$ and an error vector Δ, which in turn, in the absence of any information, is considered to be purely random in nature. We now try to estimate a weighting factor α, in the range 0.0 to 1.0, such that we can form a weighted average of two values of $\mathbf{X}$ so as to minimize the resultant Δ. That is, given:

$$\mathbf{X}_1 = \mathbf{X}_{opt} + \Delta_1 \tag{A1}$$

and

$$\mathbf{X}_2 = \mathbf{X}_{opt} + \Delta_2 \tag{A2}$$

find the value of α such that $|\Delta_3|$ is minimum, when

$$\mathbf{X}_3 = \alpha\mathbf{X}_1 + (1-\alpha)\mathbf{X}_2 = \mathbf{X}_{opt} + \Delta_3 \tag{A3}$$

It is shown in the reference that, provided the values of Δ are truly random,

$$\alpha = |\Delta_2|^2 / (|\Delta_1|^2 + |\Delta_2|^2) \tag{A4}$$

that is, the two vectors $\mathbf{X}$ should be weighted inversely as the squares of their error vectors. Clearly, this is not possible in practice, since we would need to know the value of $\mathbf{X}_{opt}$ to calculate α. However, if we assume that there is a negative correlation between the function value $f(\mathbf{X})$ and $|\Delta|$, we can at least make sure that the weighting is in the right direction by weighting the vectors directly as the function values. This is the basis of the centroid algorithm. We can continually generate random samples of $\mathbf{X}$, compute $f(\mathbf{X})$, and take a weighted mean with the current best $\mathbf{X}$, i.e. that with the highest value of $f(\mathbf{X})$ so far found. If either the random or the averaged set gives a higher merit function, this becomes the new best $\mathbf{X}$. The first value of $\mathbf{X}$ used would normally be random. It is shown in the reference that, if ideal weighting could be achieved, the rate of decrease of Δ_3 would be expected to be independent of the number of dimensions in $\mathbf{X}$.

The algorithm based on the above considerations is shown schematically in Figure A1. Some comment is required on the criterion for stopping the search. Since the search is a stochastic process, it is difficult to define a stopping criterion in terms of, for example, the derivative of the best solution with respect to iteration number.

We have therefore provided two criteria, which are (a) the exceeding of an iteration count, and (b) manual intervention by mouse or from the keyboard.

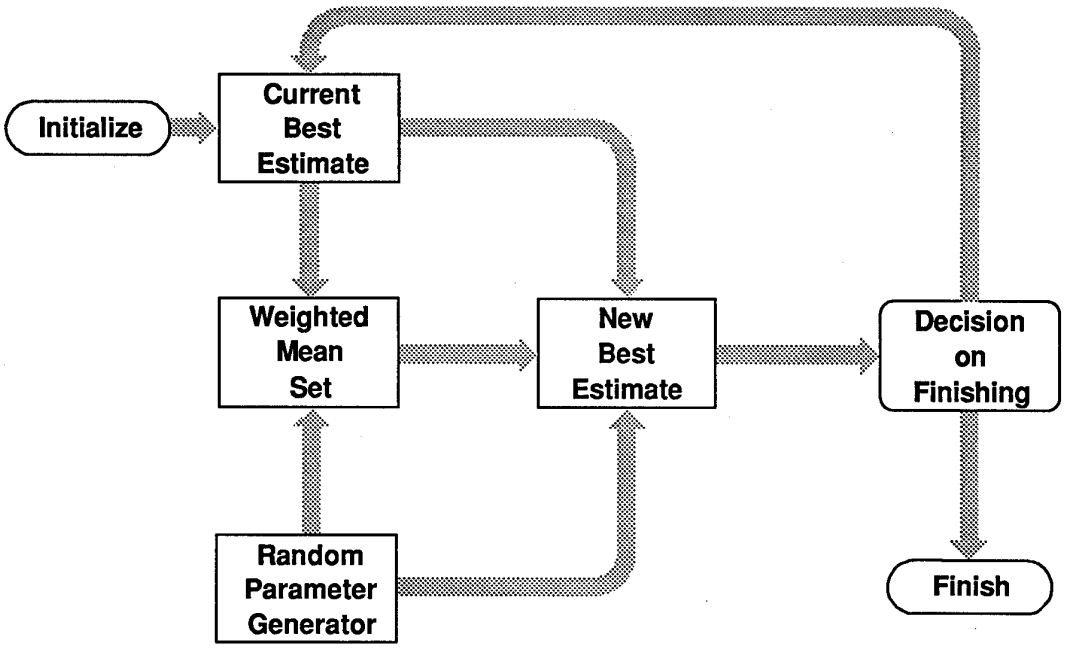


Figure A1. Schematic diagram of the centroid algorithm.

Appendix B

Sample Source Code for Dynamic-link Library

```
(****  Function for Testing Visual Basic Version of Centroid Algorithm    ****

 This is the multi-Gaussian function used in Benke & Skinner, Australian
 Computer J. vol. 23 no. 2 pp. 73-85, May  1991.

Language is Borland Pascal ver 7.0. To compile with Turbo Pascal for Windows
ver 1.5 omit the $E+ compiler directive.

 **** Author David Skinner, AMRL     ver 1.1                 November 1994 ***)

library example;        { This name is not used by programs accessing the DLL }

{$N+,E+ For access to Single type and (if necessary) emulation                 }

type
    vector = record                 { Definition of parameter set             }
                 x : Single;
                 y : Single
             end;

function MultiGaussian(var param : vector) : Single; export;
{ Definition of function to be maximized                                      }
var
        j : Word;                                   { Counts number of Gaussians }
 log_term,                                          { Logarithm of current term  }
      sum : Single;                                 { Sum of Gaussians           }
const                          { Constants used for generation of multi-Gaussian }
           a : array [1..5] of Single = ( 0.5,  1.2,  1.0,  1.0,  1.2 );
          xi : array [1..5] of Single = ( 0.0,  1.0,  0.0, -0.5,  0.0 );
          yi : array [1..5] of Single = ( 0.0,  0.0, -0.5,  0.0,  1.0 );
          si : array [1..5] of Single = ( 0.1,  0.5,  0.5,  0.5,  0.5 );
      max_log = 20;              { Maximum absolute value of parameter to Exp }
begin
    sum := 0.0;
    for j := 1 to 5 do with param do
    begin
        log_term :=  (Sqr(x - xi[j]) + Sqr(y - yi[j])) / Sqr(si[j]);
        if log_term <= max_log then sum := sum + a[j] * Exp(-log_term)
    end;
    MultiGaussian := sum
end;

exports MultiGaussian index 1;                { Making the function accessible }

begin
end.
```

SECURITY CLASSIFICATION OF THIS PAGE **UNCLASSIFIED**

REPORT NO.	AR NO.	REPORT SECURITY CLASSIFICATION
DSTO-GD-0043	AR-009-200	Unclassified

TITLE

A Windows program for multivariate optimization

AUTHOR(S)
David R. Skinner

CORPORATE AUTHOR
Aeronautical and Maritime Research Laboratory
PO Box 4331
Melbourne Victoria 3001

REPORT DATE	TASK NO.	SPONSOR
June 1995	NAV 92/315	MWSCPD

FILE NO.	REFERENCES	PAGES
G6 4/8-4286	1	21

CLASSIFICATION/LIMITATION REVIEW DATE
Not applicable.

CLASSIFICATION/RELEASE AUTHORITY
Chief, Maritime Operations Division

SECONDARY DISTRIBUTION

Approved for public release.

ANNOUNCEMENT

No limitations.

KEYWORDS

Algorithms	Optimization	Turbo Pascal (programming language)
Random variables	Variables	
Noise reduction		

ABSTRACT

A program is described that implements the "centroid" algorithm for the optimization of functions of many variables by a stochastic direct-search procedure. The program is available as a Windows application with context-sensitive on-line help, and can maximize any suitable function exported by a dynamic-link library (DLL). For practical reasons of simplicity, the program is limited to real functions of up to 20 real variables, and the search domain is limited to hyper-rectangular blocks in multi-dimensional Cartesian spaces. The only constraint on the function to be optimized is that it should be single-valued at every point in the domain, and should be capable of being evaluated in a DLL. It need not be smooth or continuous. A sample program is provided to illustrate the generation of a suitable DLL using Borland Pascal. In general, the algorithm employed will, given a sufficient number of search iterations, find the global maximum of a function in the presence of closely competing local maxima.

A Windows Program for Multivariate Optimization

David R. Skinner

DSTO-GD-0043

DISTRIBUTION LIST

Director, AMRL
Chief Maritime Operations Division
D. Richardson
Author D.R. Skinner
Dr K.K. Benke
Library, AMRL Maribyrnong
Library, AMRL Fishermens Bend
Librarian - AMRL Sydney

Chief Defence Scientist (for CDS, FASSP, ASSCM) — 1 copy only
Head, Information Centre, Defence Intelligence Organisation
OIC Technical Reports Centre, Defence Central Library
Officer in Charge, Document Exchange Centre — 8 copies
Senior Defence Scientific Adviser & Scientific Adviser - Policy and Command
Navy Scientific Adviser
Air Force Scientific Adviser, Russell Offices
Senior Librarian, Main Library DSTOS
Librarian, DSD, Kingston ACT
Serials Section (M List), Deakin University Library, Deakin University, Geelong 3217
NAPOC QWG Engineer NBCD c/- DENGRS-A, HQ Engineer Centre, Liverpool Military Area, NSW 2174
ABCA, Russell Offices, Canberra ACT 2600 — 4 copies
Librarian, Australian Defence Force Academy
Head of Staff, British Defence Research and Supply Staff (Australia)
NASA Senior Scientific Representative in Australia
INSPEC: Acquisitions Section Institution of Electrical Engineers
Head Librarian, Australian Nuclear Science and Technology Organisation
Senior Librarian, Hargrave Library, Monash University
Library - Exchange Desk, National Institute of Standards and Technology, US
Acquisition Unit (DSC-EO/GO), British Library, Boston Spa, Wetherby, Yorkshire LS23 7BQ, England
Library, Chemical Abstracts Reference Service
Engineering Societies Library, US
Documents Librarian, The Center for Research Libraries, US
Army Scientific Adviser, Russell Offices - data sheet only
Director General Force Development (Land) - data sheet only
DASD, APW2-1-OA2, Anzac Park West, Canberra ACT - data sheet only
SO (Science), HQ 1 Division, Milpo, Enoggera, Qld 4057 - data sheet only
Counsellor, Defence Science, Embassy of Australia - data sheet only
Counsellor, Defence Science, Australian High Commission - data sheet only
Scientific Adviser to DSTC Malaysia, c/- Defence Adviser - data sheet only
Scientific Adviser to MRDC Thailand, c/- Defence Attache - data sheet only

MWSCPD - Mine Warfare Systems Centre Project Director, (CMD M.S. Welford, CP2-2-19), Russell Offices, Canberra ACT 2600